Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления (ИУ)

КАФЕДРА Искусственный интеллект в системах обработки информации и управления

ДИСЦИПЛИНА Методы машинного обучения

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7

Алгоритмы Actor-Critic

название работы

Группа ИУ5-25М

Студент			Попов М.Ю.
	дата выполнения работы	*подпись*	*фамилия, и.о.*
Преподаватель			Гапанюк Ю. Е.
		подпись	*фамилия, и.о.*

Москва, 2024 г.

Цель лабораторной работы: ознакомление с базовыми методами обучения с подкреплением на основе алгоритмов Actor-Critic.

Требования к отчету:

Отчет по лабораторной работе должен содержать:

1. титульный лист;
2. описание задания;
3. текст программы;
4. экранные формы с примерами выполнения программы.

Задание:

- Реализуйте любой алгоритм семейства Actor-Critic для произвольной среды.

```
!pip install swig
!pip—install—gym[box2d]
!pip—install—pyglet
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/ Requirement already satisfied: swig in /usr/local/lib/python3.10/dist-
packages (4.1.1)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: gym[box2d] in /usr/local/lib/python3.10/dist-packages (0.25.2)
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (from gym[box2d]) (1.22.4)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from gym[box2d])
(2.2.1)
Requirement already satisfied: gym-notices>=0.0.4 in /usr/local/lib/python3.10/dist-packages (from gym[box2d])
(0.0.8)
Requirement already satisfied: box2d-py==2.3.5 in /usr/local/lib/python3.10/dist-packages (from gym[box2d]) (2.3.5)
Requirement already satisfied: pygame==2.1.0 in /usr/local/lib/python3.10/dist-packages (from gym[box2d]) (2.1.0)
Requirement already satisfied: swig==4.* in /usr/local/lib/python3.10/dist-packages (from gym[box2d]) (4.1.1)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pyglet in /usr/local/lib/python3.10/dist-packages (2.0.7)
```

```
%%bash # Install additional packages for
visualization sudo apt-get install -y python-opengl
> /dev/null 2>&1 pip install
git+https://github.com/tensorflow/docs > /dev/null
2>&1
```

```
import
collecti
ons
import
gym
import
numpy as
np
import
statisti
cs
import
tensorfl
ow as tf
import
tqdm

from matplotlib import pyplot
as plt from tensorflow.keras
import layers from typing
import Any, List, Sequence,
Tuple
```

```
# Create the
environment env =
gym.make("LunarLa
nder-v2")

# Set seed for experiment
reproducibility seed = 42
tf.random.set_seed(seed)
np.random.seed(seed)

# Small epsilon value for stabilizing
division operations eps =
np.finfo(np.float32).eps.item()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not
call `transform_   and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/gym/core.py:317: DeprecationWarning: WARN: Initializing wrapper in old step
API which retur   deprecation(
/usr/local/lib/python3.10/dist-packages/gym/wrappers/step_api_compatibility.py:39: DeprecationWarning: WARN:
Initializing environme   deprecation(
```

Модель

```
class
ActorCritic(tf.keras.Model):
"""Combined actor-critic
network."""

  def
__init__(
self,
num_actions:
int,
num_hidden_uni
ts: int):
"""Initialize.
"""
super().__init
__()

    self.common = layers.Dense(num_hidden_units,
activation="relu")     self.actor =
layers.Dense(num_actions)     self.critic =
layers.Dense(1)

  def call(self, inputs: tf.Tensor) -> Tuple[tf.Tensor, tf.Tensor]:
    x = self.common(inputs)
    return self.actor(x), self.critic(x)


num_actions = env.action_space.n  #

2 num_hidden_units = 128 model =

ActorCritic(num_actions,

num_hidden_units)
```

Сбор обучающих данных

```
# Wrap Gym's `env.step` call as an operation in a
TensorFlow function. # This would allow it to be
included in a callable TensorFlow graph.

def env_step(action: np.ndarray) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:
  """Returns state, reward and done flag given an action."""

  state, reward, done, truncated =
env.step(action)   return
(state.astype(np.float32),
np.array(reward, np.int32),
np.array(done, np.int32))


def tf_env_step(action: tf.Tensor) -> List[tf.Tensor]:
  return tf.numpy_function(env_step,
[action],
[tf.float32, tf.int32, tf.int32])


def run_episode(     initial_state: tf.Tensor,
model: tf.keras.Model,     max_steps: int) ->
Tuple[tf.Tensor, tf.Tensor, tf.Tensor]:
"""Runs a single episode to collect training
data."""

  action_probs = tf.TensorArray(dtype=tf.float32, size=0,
dynamic_size=True)   values = tf.TensorArray(dtype=tf.float32,
size=0, dynamic_size=True)   rewards =
tf.TensorArray(dtype=tf.int32, size=0, dynamic_size=True)

  initial_state_shape =
initial_state.shape   state =
initial_state

  for t in tf.range(max_steps):
```

```
    # Convert state into a batched tensor (batch
size = 1)     state = tf.expand_dims(state, 0)

    # Run the model and to get action probabilities and critic value
action_logits_t, value = model(state)

    # Sample next action from the action probability
distribution     action =
tf.random.categorical(action_logits_t, 1)[0, 0]
action_probs_t = tf.nn.softmax(action_logits_t)

    # Store critic values
values = values.write(t,
tf.squeeze(value))

    # Store log probability of the action chosen
    action_probs = action_probs.write(t, action_probs_t[0, action])

    # Apply action to the environment to get next state
and reward     state, reward, done = tf_env_step(action)
state.set_shape(initial_state_shape)

    # Store reward
rewards =
rewards.write(t,
reward)

    if tf.cast(done,
tf.bool):
break

  action_probs =
action_probs.stack()
values =
values.stack()
rewards =
rewards.stack()
    return
action_probs, values,
rewards


def get_expected_return(
rewards: tf.Tensor,     gamma:
float,     standardize: bool =
True) -> tf.Tensor:
"""Compute expected returns per
timestep."""   n =
tf.shape(rewards)[0]
  returns = tf.TensorArray(dtype=tf.float32, size=n)

  # Start from the end of `rewards` and accumulate reward sums
  # into the `returns` array
rewards = tf.cast(rewards[::-1],
dtype=tf.float32)   discounted_sum =
tf.constant(0.0)
discounted_sum_shape =
discounted_sum.shape   for i in
tf.range(n):     reward = rewards[i]
discounted_sum = reward + gamma *
discounted_sum
discounted_sum.set_shape(discounted_s
um_shape)     returns =
returns.write(i, discounted_sum)
returns = returns.stack()[::-1]

  if standardize:
    returns       =       ((returns       -

tf.math.reduce_mean(returns))            /

(tf.math.reduce_std(returns)    +    eps))

return returns
```

Actor-Critic loss

```
huber_loss = tf.keras.losses.Huber(reduction=tf.keras.losses.Reduction.SUM)
```

```
def compute_loss(
action_probs: tf.Tensor,
values: tf.Tensor,      returns:
tf.Tensor) -> tf.Tensor:
"""Computes the combined Actor-
Critic loss."""   advantage =
returns - values

  action_log_probs = tf.math.log(action_probs)

actor_loss = -tf.math.reduce_sum(action_log_probs

* advantage)   critic_loss = huber_loss(values,

returns)   return actor_loss + critic_loss
```

Функция шага обучения

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)


@tf.function def train_step(
initial_state: tf.Tensor,
model: tf.keras.Model,
optimizer:
tf.keras.optimizers.Optimizer,
gamma: float,
max_steps_per_episode: int) ->
tf.Tensor:   """Runs a model
training step."""   with
tf.GradientTape() as tape:

    # Run the model for one episode to collect
training data     action_probs, values, rewards
= run_episode(         initial_state, model,
max_steps_per_episode)

    # Calculate the expected
returns     returns =
get_expected_return(rewards,
gamma)

    # Convert training data to appropriate TF tensor
shapes     action_probs, values, returns = [
tf.expand_dims(x, 1) for x in [action_probs, values,
returns]]

    # Calculate the loss values to update
our network     loss =
compute_loss(action_probs, values,
returns)

  # Compute the gradients from the loss
grads = tape.gradient(loss,
model.trainable_variables)

  # Apply the gradients to the model's parameters

optimizer.apply_gradients(zip(grads,

model.trainable_variables))   episode_reward =

tf.math.reduce_sum(rewards)   return episode_reward
```

Цикл обучения

```
%%time

min_episodes_criterion =
100 max_episodes = 10000
max_steps_per_episode =
500

# `CartPole-v1` is considered solved if average reward is >= 475 over 500
# consecutive trials
reward_threshold =
140 running_reward =
0

# The discount factor for
future rewards gamma = 0.99

# Keep the last episodes reward
episodes_reward: collections.deque = collections.deque(maxlen=min_episodes_criterion)

t =
tqdm.trange(max_e
pisodes) for i in
t:
    initial_state = env.reset()
    initial_state = tf.constant(initial_state,
dtype=tf.float32)     episode_reward = int(train_step(
initial_state, model, optimizer, gamma,
max_steps_per_episode))

episodes_reward.append(episode_reward)
running_reward =
statistics.mean(episodes_reward)


    t.set_postfix(
episode_reward=episode_reward,
running_reward=running_reward)

    # Show the average episode reward
every 10 episodes     if i % 10 == 0:
      pass # print(f'Episode {i}: average reward: {avg_reward}')
      if running_reward > reward_threshold and i >=
min_episodes_criterion:          break print(f'\nSolved at
episode {i}: average reward: {running_reward:.2f}!')
```

```
 50%|████▉     | 4951/10000 [22:16<22:42,  3.71it/s, episode_reward=197, running_reward=140]
Solved at episode 4951: average reward: 140.12!
CPU times: user 24min 56s, sys: 29.8 s, total: 25min 26s
Wall time: 22min 16s
```

Визуализация

```
# Render an episode and save as a GIF file

from IPython import display as ipythondisplay
from PIL import Image

render_env = gym.make("LunarLander-v2", render_mode='rgb_array')

def render_episode(env: gym.Env, model: tf.keras.Model, max_steps: int):
  state = env.reset()
  state = tf.constant(state, dtype=tf.float32)
  screen = env.render()
  images = [Image.fromarray(screen[0])]

  for i in range(1, max_steps + 1):
    state = tf.expand_dims(state, 0)
    action_probs, _ = model(state)
    action = np.argmax(np.squeeze(action_probs))

    state, reward, done, truncated = env.step(action)
    state = tf.constant(state, dtype=tf.float32)

    # Render screen every 10 steps
    if i % 10 == 0:
      screen = env.render()
      images.append(Image.fromarray(screen[0]))

    if done:
      break

  return images


# Save GIF image
images = render_episode(render_env, model, max_steps_per_episode)
image_file = 'cartpole-v1.gif'
# loop=0: loop forever, duration=1: play each frame for 1ms
images[0].save(
    image_file, save_all=True, append_images=images[1:], loop=0, duration=1)


import tensorflow_docs.vis.embed as embed
embed.embed_file(image_file)
```